Cano, J., Bordallo, A., Nagarajan, V., Ramamoorthy, S. and Vijayakumar, S. (2016) Automatic Configuration of ROS Applications for Near-optimal Performance. In: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Daejeon, South Korea, 09-14 Oct 2016, pp. 2217-2223. ISBN 9781509037629.

There may be differences between this version and the published version. You are advised to consult the publisher's version if you wish to cite from it.

http://eprints.gla.ac.uk/177444/

Deposited on: 11 January 2019

Enlighten – Research publications by members of the University of Glasgow
http://eprints.gla.ac.uk

Automatic Configuration of ROS Applications for Near-Optimal Performance

José Cano, Alejandro Bordallo, Vijay Nagarajan, Subramanian Ramamoorthy and Sethu Vijayakumar
School of Informatics, The University of Edinburgh, UK

***Abstract*— The performance of a ROS application is a function of the individual performance of its constituent nodes. Since ROS nodes are typically configurable (parameterised), the specific parameter values adopted will determine the level of performance generated. In addition, ROS applications may be distributed across multiple computation devices, thus providing different options for node allocation. We address two configuration problems that the typical ROS user is confronted with: i) Determining parameter values and node allocations for maximising performance; ii) Determining node allocations for minimising hardware resources that can guarantee the desired performance. We formalise these problems with a mathematical model, a constrained form of a multiple-choice multiple knapsack problem. We propose a greedy algorithm for optimising each problem, using linear regression for predicting the performance of an individual ROS node over a continuum set of parameter combinations. We evaluate the algorithms through simulation and we validate them in a real ROS scenario, showing that the expected performance levels only deviate from the real measurements by an average of 2.5%.**

I. Introduction

ROS (Robot Operating System) [1] is a widely used framework for creating robotics software. A ROS application is a collection of software processes called *nodes*, that communicate with each other through message passing. Each node typically performs a specific task, e.g. sensing, planning, navigation, etc. A ROS node (task) is typically parameterised, where parameter values determine the content and frequency of the messages sent by the node. Therefore, parameters not only determine the performance of the node, but also the amount of computational resources required. For example, consider a ROS node implementing the navigation task of a mobile robot. By increasing the controller frequency of this task, we can increase the number of velocity commands per second sent to the robot wheels, thereby enhancing the quality (performance) of the navigation, but at the cost of increased CPU utilisation. In addition, ROS applications may be distributed, i.e. run across multiple computation devices, so nodes could be allocated to any of these devices.

Given this context, the ROS user is confronted with the complex task of configuring the ROS system as a whole in order to obtain a desired overall performance. This involves: (i) selecting the values of parameters affecting individual ROS nodes; (ii) allocating ROS nodes to computation devices. Figure 1 illustrates different configurations for a ROS system and the associated performance generated.

However, the overall performance of a ROS application is system-specific and hard to quantify in general. For example, in our case study (Section IV), performance is a function of essential requirements (e.g. avoiding collisions between agents, minimising travel time to reach target goals), as well as more sophisticated preferences (e.g. minimising close-encounters and hindrance between agents, minimising the time to infer the true agent goals). We assume that the ROS user typically has a good knowledge of the system and is able to quantify the local performance of individual ROS nodes for a given parameter value data point. Consider again a ROS node implementing the navigation task of a mobile robot. The user can quantify the positive effect on navigation upon increasing controller frequency (e.g. increases linearly up to a point and then saturates). Furthermore, the user also has good knowledge about how important the ROS nodes are in terms of how much they contribute to the overall performance. If we assume that the overall performance can be represented as the weighted sum of the individual performance of the nodes, the user can provide a good estimate of those weights.

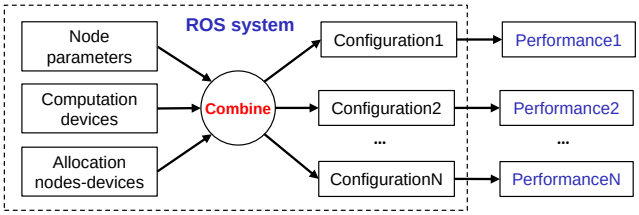

Fig. 1. Example of ROS system configurations and obtained performance.

In this paper we propose an approach[1] that allows ROS users to study and configure their systems. We first perform a characterisation of the ROS system and use linear regression analysis [2] to learn for each individual node how its performance (and resource requirement) varies as a function of its parameters. We then tackle the following two problems:

- Determining the parameter values and node allocations that maximise the overall system performance.
- Determining the node allocations that minimise the hardware required, given the parameter values.

These problems can be modelled as a constrained variant of the multiple-choice multiple knapsack problem. We provide a greedy algorithm to solve each problem, the first one uses a performance gradient, and the second one is based on the CPU requirement of the nodes. Our evaluation shows that the greedy solutions are within 1% of the optimal solution. Further evaluation on a real ROS case study validates our proposed model, with the observed performance values within 2.5% deviation of the expected ones.

[1]Code available at: `https://github.com/ipab-rad/perf_ros`

II. Problems definition

We model a general ROS system composed of N nodes and C computers. ROS nodes form a directed graph $G_n = (N,\ E)$, where pair of nodes $(n, m) \in N$ communicate through message-passing edges. An edge $e_{n,m} \in E$ is labelled with the bandwidth required, which depends on the size and frequency of the messages being sent, and is defined by a function $b : e_{n,m} \to \mathbb{R}$. Computers form an undirected graph $G_c = (C,\ L)$, where each computer $c \in C$ has a given CPU capacity defined by a function $R : c \to \mathbb{N}$. Computers can be of two types, embedded on a robot, or external — we call them *servers*. Network links between pairs of computers $(c, z) \in C$ are defined as $l_{c,z} \in L$, so that each link between computers supports one or more message-passing edges between nodes. The capacity of a link is given by its maximum bandwidth, which is defined by a function $B : l_{c,z} \to \mathbb{R}$. Depending on the type and location, computers can communicate using either wireless or wired links.

ROS nodes can have parameters, some of them are configurable and others are internal and cannot be changed. Configurable parameters generate different node settings. Thus a node $n \in N$ will be defined by a set of one or more settings, where the total number of settings depends on the type and number of parameters affecting the node. A given setting for a ROS node n_k, $k \in \mathbb{N}$, is characterised by its CPU utilisation and the performance level generated, represented by the functions $U, P : n_k \to \mathbb{R}$. We normalise the CPU utilisation of any node setting to a 'baseline' computer. We also normalise the capacity of all other computers in the system in the same way. The performance level of a ROS node is a function of the content and frequency of the messages sent. However, determining this relation automatically can be hard. Our approach assumes that a system expert manually quantifies performance levels for a small number of settings for each node. Then, we interpolate any other node setting via regression (Section IV-B).

Given the previous definitions, we model our two configuration problems as a constrained form of a multiple knapsack problem. In addition, *Problem 1* also assumes the multiple-choice generalisation — note that for *Problem 2* parameter values for nodes are given, so only one setting per node is considered. These individual problems (i.e. multiple knapsack, multiple-choice) are well-known in the literature ([3] [4]), however we consider both at the same time (for *Problem 1*) along with a set of special constraints that distinguish our formulation from previous work.

Our objective hence is to find a set of feasible allocations A of ROS nodes to computers (i.e. those that satisfy all the system constraints), and also: a) maximise the overall system performance for *Problem 1* — remember that we assume the overall performance as the weighted sum of the performance of the nodes; b) minimise the total computer capacity required for *Problem 2*. Furthermore, each node must be allocated to exactly one computer, but each computer could contain more than one node depending on its capacity. Next, we provide the mathematical description of the problems.

$$Problem1: \quad max \quad \sum_{c=1}^{|C|}\sum_{n=1}^{|N|} w_n P_{n_k} A_{cn_k} \tag{1}$$

$$Problem2: \quad min \quad \sum_{c=1}^{|C|} R_c \tag{2}$$

$$Subject\ to: \quad \sum_{n=1}^{|N|} U_{n_k} A_{cn_k} \leq R_c \quad c = 1, ..., |C|, \tag{3}$$

$$\sum_{e=1}^{|E|} b_e^l \leq B_l \quad l = 1, ..., |L|, \tag{4}$$

$$\sum_{c=1}^{|C|} A_{cn_k} = 1 \quad n = 1, ..., |N|, \tag{5}$$

$$\sum_{n=1}^{|N|} w_n = 1 \tag{6}$$

$$A_{cn_k} \in \{0, 1\} \quad \forall n, \forall c \tag{7}$$

where:

- A_{cn_k} = 1 represents that the setting k of node n has been allocated to computer c (0 otherwise).
- P_{n_k} is the performance level generated by the setting k of node n.
- w_n is the weight of node n in the overall performance (note that the value is the same for any setting).
- U_{n_k} is the CPU utilisation of the setting k of node n.
- R_c is the CPU capacity of computer c.
- b_e^l is the bandwidth required by edge e, which is supported by network link l.
- B_l is the maximum bandwidth of network link l.

The first set of constraints (3) ensures that the nodes allocated to a computer do not exceed its capacity. The second set of constraints (4) guarantees that the bandwidth of any network link is not exceeded. The third set of constraints (5) ensures that every node is allocated to exactly one computer — note that since a ROS node cannot assume two different settings at the same time, it is not necessary to add an extra restriction to guarantee that exactly one setting of each node is allocated. The last set of constraints (6) ensures that the overall value for any combination of weights for the nodes is always the same. Finally, we add two new sets of constraints to the model, which specifically apply to distributed ROS systems.

Residence constraints: restrict the particular subset of computers $C' \subset C$, to which a given node n may be allocated. This makes sense, for example, when nodes are directly connected to sensors/actuators on a given robot.

$$n \in N \ \wedge \ c \in C' \implies A_{cn_k} = 1 \tag{8}$$

Coresidence constraints: restrict the subset of valid allocations such that pairs of nodes (n, m) must always reside on the same computer. In practice, this may be required when the long latency of a network link is not tolerable.

$$n, m \in N \wedge c, z \in C : (A_{cn_k}, A_{zm_q}) \implies c = z \tag{9}$$

III. Algorithmic solutions

We now describe the two greedy algorithms that solve the optimisation problems proposed. Both algorithms provide near-optimal solutions (see Section V-B). In addition, the solutions found are always feasible (i.e. satisfy all the constraints) for any ROS system. However, finding solutions may depend on the specific constraints of each system.

A. Problem 1: maximising performance

The first greedy algorithm uses a heuristic based on the performance gradient, $\vec{\nabla}P$, of the configurable nodes (i.e. those with configurable parameters). We assume that there are $M <= N$ configurable nodes. Each point in the gradient vector is determined by the CPU utilisation of the nodes, $\vec{\nabla}P(U_1, ..., U_m)$, and the value for each point is given by the best relative increment in performance for a unit of CPU utilisation — note that the performance level corresponding to each CPU utilisation value can be obtained by applying regression analysis (Section IV-B). The procedure is described in Algorithm 1 and consists of two parts: i) an initial allocation of nodes that satisfies the system constraints; ii) an allocation refinement that attempts to maximise the overall performance by relocating nodes and updating configurable nodes using the performance gradient, when possible.

Algorithm 1 Greedy heuristic Problem 1

```
1:  while BW_constraints satisfied do
2:      A ← allocate nodes with Res_constraints
3:      A ← allocate nodes with CoRes_constraints
4:      A ← allocate remaining nodes
5:  if BW_constraint ¬satisfied then
6:      return
7:  N_conf = select configurable nodes from N
8:  UPGRADE_CONF_NODES(A, N_conf)
9:  C_full = select computers from C where R_c.free == 0
10: for c in C_full do
11:     A ← move any n with U_{n_k} < U_{n_max} to C̄_full
12: UPGRADE_CONF_NODES(A, N_conf)
13: Update C_full
14: for c in C_full do
15:     A ← move any n to C̄_full
16: UPGRADE_CONF_NODES(A, N_conf)
17: return A
```

The initial allocation (lines 1-6) assumes that: i) configurable parameters are set to their minimum values, thus generating the lowest CPU utilisation and performance level; ii) the ROS system is able to work with this configuration; iii) R_c is fixed for all computers. Then, nodes with residency (*Res*) constraints are allocated to the corresponding computers. Next, nodes with coresidency (*CoRes*) constraints are allocated, prioritising nodes with highest CPU utilisation and computers with largest capacity. Finally, the remaining nodes are allocated using the same prioritisation policy. Note that if all the bandwidth (*BW*) constraints are satisfied, the allocation order (outline above) always guarantees a solution.

Algorithm 2 UPGRADE_CONF_NODES(A, N_{conf})

```
1:  V_aux = []
2:  for n in N_conf do
3:      if U_{n_k} < U_{n_max} then
4:          V_aux.add(n)
5:  while V_aux ≠ [] do
6:      U_{n_k} = ∇⃗P(U_1, ..., U_m)
7:      if U_{n_k} < U_{n_max} then
8:          c = A(n)
9:          if R_c.free > 0 then
10:             U_{n_k}+=1
11:         else
12:             V_aux.del(n)
13:     else
14:         V_aux.del(n)
15: return
```

In allocation refinement (lines 7-16), first configurable nodes are upgraded following Algorithm 2, which selects nodes based on the performance gradient (note that $c = A(n)$ gets the currently allocated computer of node n) and increases the CPU utilisation of the currently selected node by one unit in each iteration. The process stops when no more increments are possible, because nodes reached their maximum CPU utilisation ($U_{n_{max}}$, which can be obtained via regression) or computers reached their maximum capacity — R_c.free is the current free capacity of computer c. Then (lines 9-12, Algorithm 1) nodes that did not reach their maximum utilisation allocated to computers that reached the maximum capacity (we called them full computers, C_{full}) are moved to computers that did not ($\overline{C}_{full}$). Algorithm 2 is called again to fill up the new CPU capacity generated. Finally (lines 13-16, Algorithm 1), the set of full computers is updated and any node from full computers, having reached its maximum CPU utilisation or not, is moved to a computer with enough free capacity to contain it. Algorithm 2 is called for the last time, possibly allowing to further improve the overall performance.

Note that to move nodes, selections are also made according to the gradient. Furthermore, computers are selected by maximum capacity and always guaranteeing that new allocations do not violate any previously satisfied constraints.

B. Problem 2: minimising computer capacity

The second greedy algorithm uses a simple heuristic that attempts to allocate nodes with highest CPU utilisation to computers with lowest capacity first (it is based on previous work [5]). In addition, it assumes that the initial capacity of any $server$ in the system is 0, thus being increased when required. The procedure is described in Algorithm 3.

Initially nodes with residency constraints are allocated. Since residency constraints may imply running nodes in computers whose capacity cannot be increased (e.g. robot's on-board computer), we allocate these nodes first to guarantee their allocation. Then the remaining nodes are allocated, $A(n) = c$, following the described heuristic while satisfying

Algorithm 3 Greedy heuristic Problem 2

1: N_{max} = sort $nodes$ by max CPU_utilisation
2: C_{min} = sort $computers$ by min capacity
3: $A \leftarrow$ allocate $nodes$ with Res_constraints
4: **for** n in N_{max} **do**
5: **for** c in C_{min} **do**
6: **if** n satisfies CoRes_constraints in c **then**
7: **if** R_c.free $>= U_{n_k}$ **then**
8: $A \leftarrow A(n) = c$
9: **else if** c.type == server **then**
10: R_c += $U_{n_k} - R_c$.free
11: $A \leftarrow A(n) = c$
12: **if** $A(n)$ == NULL or BW_constraint ¬satisfied **then**
13: **return**
14: **return** A

coresidency constraints. If at some point the selected computer is a $server$ and cannot allocate the currently selected node, its capacity is increased to exactly satisfy the required CPU utilisation for the node (line 10). Note that if all the bandwidth constraints are satisfied, finding solutions only depends on the coresidency constraints of the system.

IV. Case study

We now present a real ROS distributed system that is a particular instantiation of the general model presented in Section II. The system is composed of two types of agents: autonomous robots with on-board processing and sensing capabilities; and humans. Each agent is pursuing a goal (i.e. a spatial position in the scenario) while avoiding collisions with other agents. In addition, an external server has access to network cameras which can track people inside the environment. Robots infer the goals and future motion of other agents using tracking data and online sensor processing (see [6] for more info). Server and robots communicate wirelessly whereas network cameras and server connect through Ethernet, thus defining the network graph G_c.

Figure 2 shows the node graph G_n of the case study, where multiple ROS nodes are interconnected through ROS *topics* and *services*. Some nodes within the robot namespace may run on the server, thus potentially improving the overall performance. In addition, edges between nodes are labelled with the expected range of message frequencies, which can be easily translated to the required bandwidth.

We now describe briefly each ROS node, highlighting those with critical parameters (one per node) that can generate different settings, thus modifying their performance.

Parameterised Nodes:

- *Tracker*: One instance per network camera that forms part of a distributed person tracking algorithm [7]. The critical parameter is the output frame rate. The higher the frame rate, the more accurate the tracking.
- *Model*: Provides intention-aware predictions for future motion of interactively navigating agents, both robots and humans. A higher number of modelled agent goals will lead to more accurate goal estimates.

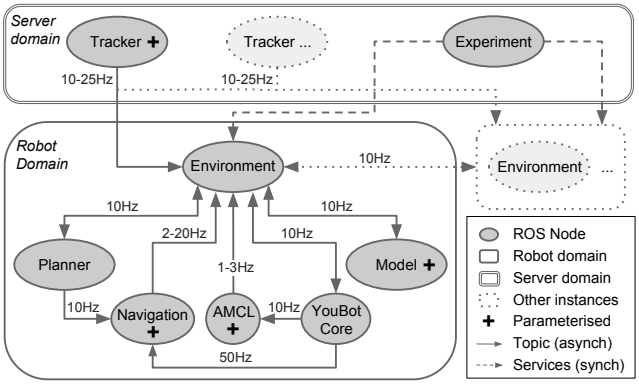

Fig. 2. Case study: Node graph, G_n, composed of one *Experiment* node, one *Tracker* node per network camera, and six nodes per robot.

- *AMCL*: The Adaptive Monte Carlo Localisation [8] relies on laser data and a known map of the environment. The number of particles the algorithm may use during navigation defines the localisation robustness.
- *Navigation*: Avoids detected obstacles and plans a path given a costmap, finally producing the output velocity the robot must take. The higher the controller frequency, the more reactive and smooth the navigation is.

Non-Parameterised Nodes:

- *Experiment*: A server node that basically coordinates all robots taking part in the experiment.
- *Environment*: Combines information generated by the local robot, other robots or other nodes (i.e. *Tracker*).
- *Planner*: Generates a navigation costmap (used by the *Navigation* node) that encodes the future motion of all agents with respect to other agents' motion given their inferred target goals from the *Model*.
- *YouBot_Core*: A set of ROS packages and nodes (e.g. etherCAT motor connectivity, internal kinematic transformations, interface with the laser scanner, etc) that enables the robots (KUKA youBots) to function.

A. System characterisation

In order to test our algorithmic solutions, we characterised each node in Figure 2 using common monitoring tools from Linux (e.g. *htop*) and ROS (e.g. *rqt*). Table I summarises the measured values. Columns two and three show residence and coresidence constraints. Column four shows the settings selected by the system expert for each configurable node. The next three columns show the average values of CPU utilisation, message frequency and bandwidth required for each node setting — note that there is only one setting for non-parameterised nodes. The last two columns represent the performance level for each node setting and the weight (w_n) of each node, both quantified by the system expert.

The robots' on-board computers are 1.6GHz Intel Atom dual core with 2GB RAM. The server used is a 3.30GHz Intel i5 quad core with 16GB RAM. All the CPU measurements are normalised to the robot CPU capacity, with a value of 100. The server capacity was estimated based on the results provided by SPEC CPU2006 [9]. The networks employed are a wireless 802.11ac at 300Mbps, and a 1Gbps Ethernet.

TABLE I

ROS NODES CHARACTERISATION

Node	Res	CoRes	Parameters	CPU util.	Freq (Hz)	BW (KB/s)	Performance	Weight
Experiment	server	-	-	1	10	1	100	0.05
Tracker	server	-	Output freq: 10 15 20 25	80 120 160 200	10 15 20 25	1.0 1.5 2.0 2.5	40 70 90 100	0.2
Environment	-	-	-	1	10	0.5	100	0.05
Model	-	-	Num. goals: 4 3500 10000	17 40 60	10 10 10	5 5 5	20 70 100	0.2
Planner	-	Navigation	-	1	10	0.5	100	0.05
AMCL	-	-	Particles: 200 500 3000	19 41 66	2.5 2.5 2.5	1 1 1	20 50 100	0.2
Navigation	-	Planner	Controller freq: 2 10 20	25 39 50	2 10 20	0.1 0.5 1.0	10 65 100	0.2
Youbot_Core	robot	-	-	16	10	0.5	100	0.05

B. Regression analysis

Given the values for the parameter configurations (node settings), CPU utilisation and performance level in Table I, we obtained the linear regression graphs relating them for the four configurable nodes: *Tracker*, *Model*, *AMCL* and *Navigation*. Figure 3 shows the relationship between the parameter values and the performance level generated, and Figure 4 between CPU utilisation and performance.

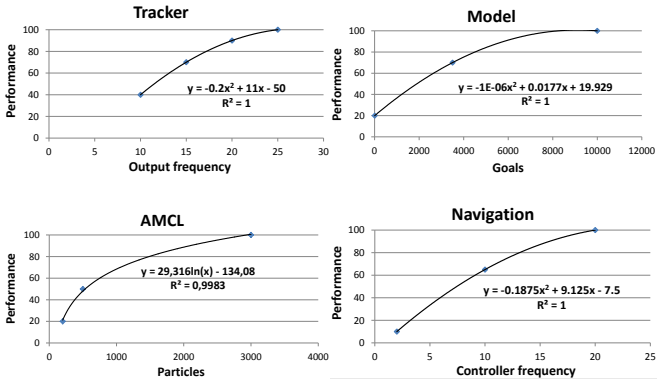

Fig. 3. Linear regression graphs: parameter values vs performance level.

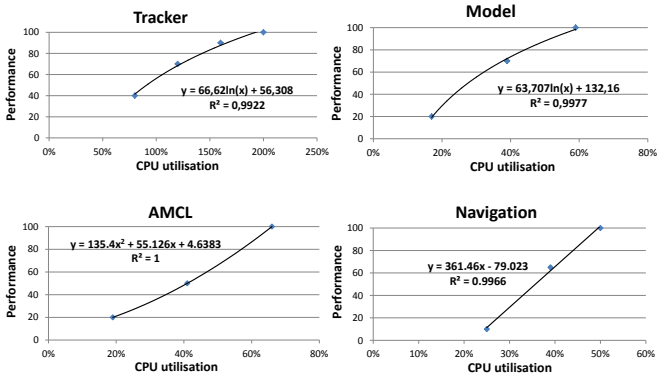

Fig. 4. Linear regression graphs: CPU utilisation vs performance level.

Note that all the regression graphs are based on four samples at most, which we find to be sufficient as the regression coefficients (R^2) are "1" or close enough to "1", meaning that the selected samples fit well in a curve. Furthermore, in the experiments reported (Section V-C), the solutions provided by simulation, which are based on predicted values obtained via regression, match the behaviour of the the real system, validating the accuracy of the predictions. However, other ROS systems may require more samples to obtain a regression model that fits the real system properly.

The regression equations in Figure 4 are used in *Problem 1* to obtain the performance gradient. In *Problem 2*, we also use the graph relating the parameter values and CPU utilisation — this and other regression graphs are not shown but are available on the online repository. Finally, it is worth noting that assigning different performance levels for the nodes in Table I will generate different regression graphs, which in turn might affect the allocation solutions provided in both problems. Our methodology helps the user to explore different combinations in order to make the best choice.

V. EVALUATION

In this section, we first define a set of system instances of increasing size derived from the case study presented. Then, we evaluate our algorithmic solutions with the objective of answering the following research questions:

- *RQ1*: How well do our greedy heuristics perform on the system instances compared to the optimal solutions?
- *RQ2*: How well do the ideal allocation solutions provided by our two algorithms translate into real configurations of the case study?
- *RQ3*: How would the real system behave if we modify the parameter values provided by our algorithms?

A. System instances

In order to obtain different instances of our case study, we only need to add robots and/or cameras to the baseline system (i.e. one robot, one camera), as dashed lines show in Figure 2. Increasing these elements, the total number of ROS nodes will change accordingly, thus producing more challenging problems. Table II summarises the set of instances analysed, including the total number of nodes (Nodes) and configurable nodes (cNodes) present in each case.

TABLE II

SYSTEM INSTANCES CONSIDERED

Instance	Computers	Robots	Cameras	Nodes	cNodes
1	2	1	1	8	4
2	2	1	2	9	5
3	2	1	3	10	6
4	3	2	1	14	7
5	3	2	2	15	8
6	3	2	3	16	9

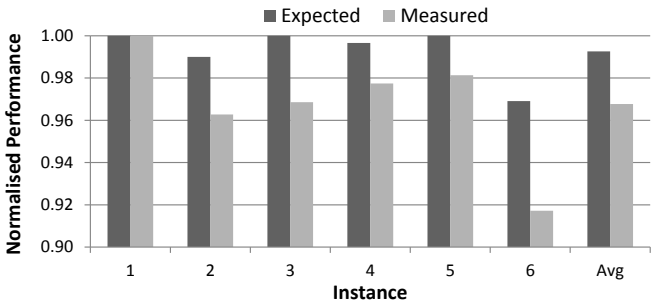

Fig. 5. Problem 1: Comparison between expected and measured overall performance. All values are normalised to the optimal solution (value = 1).

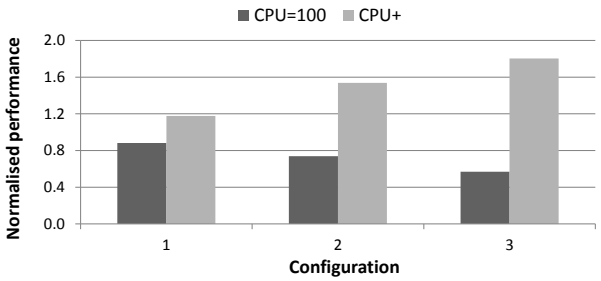

Fig. 6. Problem 1, Instance 1: Comparison of different node settings when not increasing (CPU=100) and when increasing (CPU+) the robot's capacity. Values are normalised to the baseline configuration (performance = 1).

B. Simulation: algorithms analysis

We first analyse the solutions provided by our two greedy algorithms comparing them with the corresponding optimal solutions for the instances described in Table II. Remember that optimal solutions will be given by allocations of node settings to computers that: i) maximise the overall performance for *Problem 1* — note that individual performance values for a given node setting are obtained by applying the regression equations in Figure 3; ii) minimise the total CPU capacity required for *Problem 2* — note that in this case we set the maximum value for each node parameter, which translates into the maximum performance (and CPU requirement) possible. The optimal solution for each case was obtained by executing brute force algorithms, which required several hours to complete for some instances.

Answering *RQ1*, we found that both heuristics provide near-optimal solutions for all the instances analysed. Figure 5 shows the results for *Problem 1*, where values for the greedy heuristic (*Expected*) are normalised to the optimal ones (value "1" for all the instances). As it can be seen, the difference with the optimal solution for *Problem 1* is less than 1% on average. For *Problem 2*, the average difference is less than 0.1% — the differences being negligible, they are not shown in a graph.

C. Case study: behaviour analysis

Given the previous results, we now compare the behaviour of the case study with the expected values. For each instance in Table II, we configure the ROS nodes in the real system with the parameter settings and the specific allocation provided by the two algorithmic solutions. Then, we check if the real system matches a specific configuration by monitoring the frequencies of the messages sent by the ROS nodes. If the observed frequency values are close to the expected ones (obtained from Table I or by regression analysis) for all the nodes, the real system matches the given configuration. Otherwise, the expected/observed frequencies can differ due to: a) overloaded computers; b) approximation errors in the system characterisation and/or regression analysis.

In our case, the observed frequencies for all the instances analysed and for both problems deviate less than 3% on average from the expected ones, which answers *RQ2* and validates our approach, that is, the accuracy of the system characterisation and the regression analysis performed.

Furthermore, given the observed frequencies, we estimate the real system performance for the nodes, $P_{measured}$, by applying the following formula:

$$P_{measured} = P_{expected} \times \frac{F_{measured}}{F_{expected}} \tag{10}$$

where $P_{expected}$ is the expected performance predicted by our algorithms, $F_{expected}$ is the expected frequency for each ROS node and $F_{measured}$ is the observed frequency during the experiment. For *Problem 1*, results are also included in Figure 5 (*Measured*), where measured performance values only deviate by 2.5% on average from the expected ones. For *Problem 2* we obtained similar results (not shown).

Finally, we analyse the effect on the case study behaviour when increasing some of the parameter values provided by our algorithms. In particular, we consider several node settings for *Instance 1* when nodes in the robot domain are forced to run in the robot. Table III shows the configuration provided by Algorithm 1 (*Baseline*) and the three modifications considered; only configurable nodes are shown.

TABLE III
PARAMETER SETTINGS CONSIDERED FOR INSTANCE 1

Configuration	Model (goals)	AMCL (particles)	Nav. (Hz)
Baseline	80	200	15
Mod. 1	200	300	16
Mod. 2	3500	1000	18
Mod. 3	10000	3000	20

Figure 6 shows the results, where *CPU=100* means that the robot capacity does not change. Recall that increasing the parameter values causes the CPU utilisation of each node to increase. For *CPU=100*, this translates into an overloaded computer, even for the first modification (*Mod. 1*). Answering *RQ3*, this parameter modification leads to the CPU capacity constraint not being satisfied, which degrades performance, further validating our approach. The figure also shows the performance improvement if the CPU capacity were to be increased (*CPU+*) as required.

VI. DISCUSSION

Finally, we briefly discuss two important issues referred to previously: i) the quantification of individual performance for the nodes; ii) the relationship between the individual performance of the nodes and the overall system performance.

In the first case, since a ROS node could have a complex relationship between the parameter values and performance generated, instead of giving the ROS user the responsibility of assigning performance levels, it might be possible to consider developing methods to automate the process ([10], [11]). However, applying such methods is orthogonal to our proposal which we reserve for future work.

In the second case, we have assumed a linear contribution of the individual performance of each node to the overall value (expressed via node weights), which seems to work well for our case study. However, this relationship might be non-linear for more complex systems. In such a case, it might be possible to perform a sensitivity analysis [12] based on the individual contributions to determine the relationship more accurately, but we leave it for future work.

VII. Related work

This work is a generalisation of our previous proposal [13]. Whereas we address system configuration over a continuum set of parameter combinations, the previous work only assumes a small number of parameter configurations (called variants). This important consideration offers much more flexibility to the ROS user and has increased the efficacy of our greedy solutions, bringing them closer to optimal.

There are many other prior works addressing task allocation in distributed robotics. A comprehensive taxonomy can be found in [14], where problems are categorised based on: i) the degree of interdependence of agent-task utilities; and ii) the system configuration, which in turn is based on an earlier taxonomy [15]. According to these taxonomies, the two problems discussed in this paper fall in the category of Cross-schedule Dependencies (XD). Other works based on the linear assignment problem [16] assume a single task per agent [17], [18], [19]. In our case, the number of tasks is equal or greater than the number of agents. In [20], [21] several agents are needed to complete each task, which is a subset of our problem. Finally, in [22] heterogeneous tasks and multiple instances for each task are assumed, but it does not consider different configurations of the same task.

To summarise, no previous work in robotics addresses all the following considerations: a constrained, distributed, heterogeneous system with more tasks than agents and a continuum set of different configurations for the tasks.

VIII. Conclusions

We have proposed an approach for automatically configuring ROS applications. The approach is based on performing a system characterisation and applying linear regression to get the configuration that can optimise the system, either maximising performance or minimising the hardware resources required. We have modelled these optimisation problems mathematically and we have proposed two greedy algorithms to solve them, whose solutions deviate from the optimal by less than 1% on average. We have validated our algorithms in a real ROS environment, observing an average difference between estimated and measured performance of 2.5%.

Acknowledgments

This work was supported by the AnyScale Applications project under the EPSRC grant EP/L000725/1, and partially by the EPSRC grant EP/F500385/1 and the BBSRC grant BB/F529254/1.

References

[1] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

[2] X. Yan and X. G. Su, *Linear Regression Analysis: Theory and Computing*. World Scientific Publishing Co., Inc., 2009.

[3] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack problems*. Springer, 2004.

[4] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., 1990.

[5] J. Cano, E. Molinos, V. Nagarajan, and S. Vijayakumar, "Dynamic process migration in heterogeneous ROS-based environments," in *17th International Conference on Advanced Robotics (ICAR)*, July 2015.

[6] A. Bordallo, F. Previtali, N. Nardelli, and S. Ramamoorthy, "Counterfactual reasoning about intent for interactive navigation in dynamic environments," in *IEEE/RSJ International Conference on IntIntelligent Robots and Systems (IROS)*, October 2015.

[7] F. Previtali and L. Iocchi, "PTracking: distributed multi-agent multi-object tracking through multi-clustered particle filtering," in *International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, Sept 2015.

[8] P. Pfaff, W. Burgard, and D. Fox, "Robust monte-carlo localization using adaptive likelihood models," in *EUROS*, 2006.

[9] J. L. Henning, "SPEC CPU2006 Benchmark Descriptions," *SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, Sept. 2006.

[10] C. G. Cassandras, Y. Wardi, B. Melamed, G. Sun, and C. G. Panayiotou, "Perturbation analysis for online control and optimization of stochastic fluid models," *IEEE Transactions on Automatic Control*, vol. 47, no. 8, pp. 1234–1248, Aug 2002.

[11] Y.-C. Ho and X.-R. Cao, *Perturbation analysis of discrete event dynamic systems*, ser. The Kluwer international series in engineering and computer science. Boston: Kluwer Academic Publishers, 1991.

[12] A. Saltelli, M. Ratto, T. Andres, F. Campolongo, J. Cariboni, D. Gatelli, M. Saisana, and S. Tarantola, *Global Sensitivity Analysis: The Primer*. Wiley, 2008.

[13] J. Cano, D. R. White, A. Bordallo, C. McCreesh, P. Prosser, J. Singer, and V. Nagarajan, "Task variant allocation in distributed robotics," in *Proceedings of Robotics: Science and Systems (RSS)*, June 2016.

[14] G. A. Korsah, A. Stentz, and M. B. Dias, "A comprehensive taxonomy for multi-robot task allocation," *The International Journal of Robotics Research*, vol. 32, no. 12, pp. 1495–1512, Oct. 2013.

[15] B. P. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *The International Journal of Robotics Research*, vol. 23, no. 9, pp. 939–954, 2004.

[16] D. W. Pentico, "Assignment problems: A golden anniversary survey," *European Journal of Operational Research*, vol. 176, no. 2, 2007.

[17] C. Nam and D. Shell, "Assignment algorithms for modeling resource contention and interference in multi-robot task-allocation," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, May 2014.

[18] L. Luo, N. Chakraborty, and K. Sycara, "Provably-good distributed algorithm for constrained multi-robot task assignment for grouped tasks," *Robotics, IEEE Transactions on*, Feb 2015.

[19] L. Liu and D. A. Shell, "Optimal market-based multi-robot task allocation via strategic pricing," in *Proceedings of Robotics: Science and Systems*, Berlin, Germany, June 2013.

[20] J. Chen, X. Yan, H. Chen, and D. Sun, "Resource constrained multirobot task allocation with a leader-follower coalition method," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, Oct 2010.

[21] Y. Zhang and L. E. Parker, "Considering inter-task resource constraints in task allocation." *Autonomous Agents and Multi-Agent Systems*, vol. 26, no. 3, pp. 389–419, 2013.

[22] N. Miyata, J. Ota, T. Arai, and H. Asama, "Cooperative transport by multiple mobile robots in unknown static environments associated with real-time task assignment." *IEEE T. Robotics and Automation*, vol. 18, no. 5, pp. 769–780, 2002.

View publication stats